2

Automated Modeling of Physical Systems

The modeling activity can be seen as the process of organizing knowledge about a system. This activity is of major interest in every single scientific and engineering discipline. It makes possible to do essential assessment before systems are built, it can alleviate the need for expensive experiments and it can provide support in all stages of a project from conceptual design through commissioning and operations. Unfortunately, the modeling process usually becomes a bottleneck in many scientific or engineering endeavors. Over the last decade different approaches have been investigated in order to automate the modeling process minimizing its cost. The decomposition of models into submodels, in a similar way as the system is composed by subsystems, is an approach widely adopted by most of the present modeling tools (different approaches adopted by some modeling environments are discussed in this chapter).

As it has been discussed in the previous chapter, achieving the simulation model of the whole system is not merely a question of connecting its component models. The simulation model represents the behaviour of a physical system according to some experimentation purpose (see Definition 1.12). However, the same system component may require different simulation models according to its operation contexts. For instance, observe the physical device in Figure 2.4. This device may operate as a motor if a voltage source is connected to the armature circuit. The same device can be used as current generator if a torque is applied to its shaft. Whether the simulation model of the electro-mechanical device should represent the motor or the generator behaviour is a decision that can not be taken until the model reusing context is defined.

If different simulation models are provided as predefined modeling components for the same physical device, the model user is involved in the procedure of deciding which of them has the adequate formulation for each particular reusing context. Hence, the user is involved in the implementation details (model formulation), and this is just what a modeling tool based on model reusability tries to avoid.

The alternative is to provide the user with a unique model which can be manipulated in order to obtain the simulation model according to the reusing context. This is the goal of the object-oriented modeling methodology: allow the user to specify the system at one of the structure levels (see Table 1.1) and manipulate that specification to adapt its mathematical formulation into the reusing context according to the experimentation purpose by considering both the experimental framework and the adequacy level.

The object oriented modeling methodology has several analogies with respect to the object-oriented programming. Languages such as *Smalltalk* (Goldberg 1983) or *C++* (Stroustrup 1987) popularized a paradigm where the cost of software development is minimized by structuring the code in a modular way, allowing the reuse of objects predefined in libraries. These objects encapsulate software behaviour and may be reused without going into their implementation details. The benefits of the object oriented methodology as a way of organizing knowledge are widely accepted (see for instance (Rumbaugh *et al.* 1991)). The advantages of using this paradigm as the framework for the modeling process of dynamic systems have been thoroughly investigated in different engineering disciplines as chemical process (e.g. (Nilsson 1993, Urquía 2000)), power systems (e.g. (Mattsson 1992, Glaser *et al.* 1995)) or mechatronics (e.g. (Hahn 1995)).

Despite of the analogies between object-oriented programming and modeling domains, significant differences also exist between them. The essential aspects of these differences, to be discussed in the next chapter (see Section 3.2), will constrain the way in which physics behaviour should be modularized and represented. PML is a modeling language designed to overcome these limitations. This chapter introduces the basic concepts of the Object-Oriented paradigm and its adaptation to the system behaviour modeling techniques. Many of the essential aspects of present modeling tools have been incorporated in PML. Hence, some of these modeling environments will be briefly introduced, giving an overview of the different approaches, with the aim of analyzing their modeling constructs in order to uphold the basis of the PML modeling framework.

2.1. Automated modeling

The amount of effort required to establish an appropriate simulation model of a production process prevents the broad use of simulation techniques in industrial practice. All the tasks involved to set up a reliable model, from compiling all the required knowledge (phenomena and laws, property tables, etc.) to model parameterization and validation[1], are time consuming and error prone. Consequently, the need of computer tools to automate the modeling process is being claimed since the early eighties, as for instance (Sargent 1983, Marquardt 1991, Mattsson 1993, Cellier and Elmqvist 1993, Marquardt 1996, Piera *et al.* 1998).

The main objective of the methods and tools to be discussed in this section is to eliminate the perceived modeling bottleneck by providing with tools able to offer effective assistance to the modeling process. These modeling tools should offer the mechanisms that the user needs to describe a system in terms of modeling elements as familiar to him as possible such as circuit diagrams or process flowsheets (see Figure 2.1). The modeling tool should also automate the procedure required to achieve the model simulation results. Hence, two main task should be covered by a modeling and simulation environment: provide with a modeling methodology able to minimize the system model construction burden and allow the experimentation task to be performed through the built model.

Figure 2.2 illustrates an ideal situation: the user builds and operates with a model which

[1]Model parameterization and validation are out of the thesis scope.

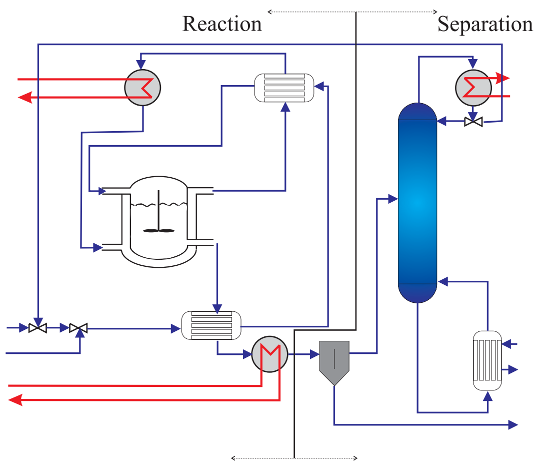


Figure 2.1. Topological representation of a chemical plant by means of the process unit flowsheet

is almost a picture of the system (e.g. the process flowsheet), and the user can obtain the results according to his experimentation purposes. All the mathematical aspects related to the simulation, such as algebraic and computational manipulations, remain hidden.

To automate the modeling and simulation procedure has been a common goal of the different methodologies. However, there is still very much work to do since, as it is discussed through this chapter, each approach shows different limitations that do not allow to consider the modeling and simulation procedure being completely automated.

The capability to reuse predefined models has been shown as a solution to simplify the modeling process. The model of a complex system can be recursively decomposed into submodels representing the system components. This decomposition process is illustrated in Figure 2.3. Basic component models are based on fundamental physical laws obtained from first principles as, for instance, the conservation laws for energy, mass or momentum.

By allowing model decomposition in ever simple submodels, the synthesis of complex models could be reduced to the specification of its structure, i.e., its components and their connections. A desirable property of such structure would be its correspondence to the system topology in order to simplify the modeling task by keeping a close analogy model $\Longleftrightarrow$ system.

However, as it has been discussed in Chapter 1 model formulation at a structure level rarely

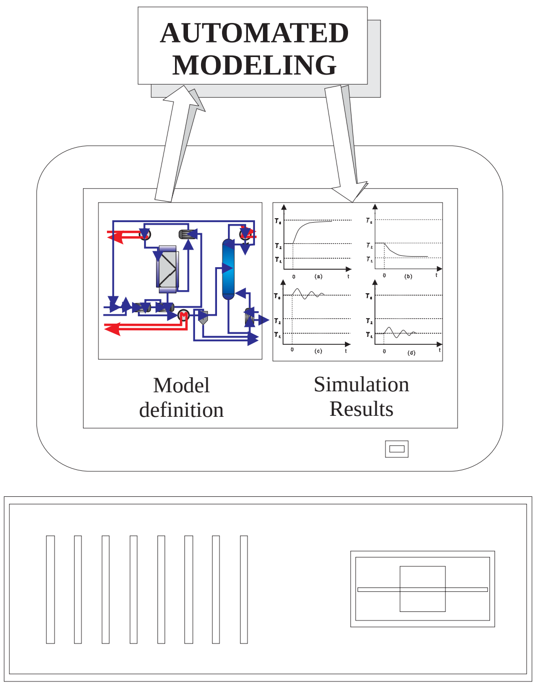


Figure 2.2. Automated modeling-simulation process

matches the formulation suitable to solve the model numerically. Therefore, different manipulations have to be performed in order to adapt the structured model into a system of equations with an adequate computational structure (the simulation model). Frequently this translation procedure requires of the user supervision and intervention. Nevertheless, this is not the only point where the user is involved with problems which are not related with his purposes, i.e, the experimentation.

The level of automated modeling achieved by the different methodologies depends on several factors. Hence, it is very important to establish a set of features to measure the level of automation in order to compare the different modeling methodologies. The following criteria will be considered in this thesis:

- **Attending to model construction**. The assistance facilities provided to define new models at the topological structure level:
  - By means of aggregation predefined models can be reused and connected to build up

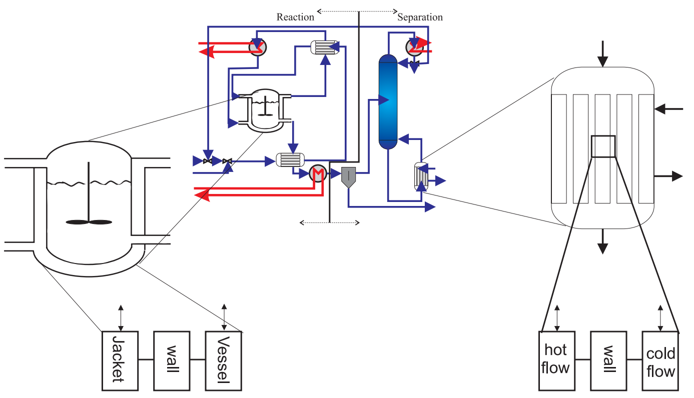


Figure 2.3. Structural decomposition of a chemical plant.

a new model. The connections of aggregated models should preserve the analogy with the physical system (for instance, if two system components exchange matter through a connection point, their counterpart models should define the analogous connection defining the exchange of matter).

– New models can also be build up from the first principles.

In both cases the user should easily identify the physical behaviour represented in a modeling component. So the formalism used to declare the system behaviour should be close to the physical knowledge the modeller has about the system. The discussion of these factors in the PML modeling environment is analyzed in Chapters 3 and 4.

- **Attending to model usage**. A ready-made model can be adapted to different simulation purposes. According to the methodology proposed in Chapter 1, model adaptation refers to the capability to translate the system representation at one of the structure levels (topological or functional model) into the specification at the causal explanation level (simulation model). It will introduce the following demands to the modeling environment:

  – The model manipulation tools provided to adapt a predefined model to different simulation purposes (control system design, operator training, ...). These tools basically consist in the capability to postulate hypothesis in order to achieve simulation models having a complexity according to the experimentation objective.

- Simulation model generation. The capability of a modeling methodology to provide the means to shield the user from all the work which has to be done to translate the system representation at the non causal specification level (mathematical model) into the specification at the causal explanation level (simulation model). This task usually involves algebraic and symbolic manipulations that should remain hidden to the end user.

The user will not be involved in the manipulation of the structured models. This is not merely a question of having a proper development methodology, as for instance the object oriented method, with a representation formalism suitable to express the physical behaviour. It is also necessary to impose to the representation formalism the capabilities to permit a maximal separation between the model formulation and its problem solving purpose, thus, the same model may be used for different experimentation purposes and all the required model manipulations would be performed automatically by the modeling environment. The contributions of PML in this aspect will be presented in Chapter 5.

2.2. Modeling methodologies

This section introduces the most extended methodologies in continuous system modeling. Several of them, e.g. continuous system simulation (CSSL standard (Strauss 1967)) and block oriented tools, are considered as classical nowadays, even they are still in use. The discussion is addressed to highlight the limitations of these tools in relation with the automated modeling objective.

The model of the device shown in Figure 2.4 will be used to illustrate the constraints imposed to the reusability by the modeling methodologies currently considered as classical. The system model has been extracted from (Cellier 1991) and it has been frequently used in the literature to illustrate the problems to reuse predefined models shown by different modeling methodologies.

The electric device model will be derived from the first principles, i.e., using the laws of

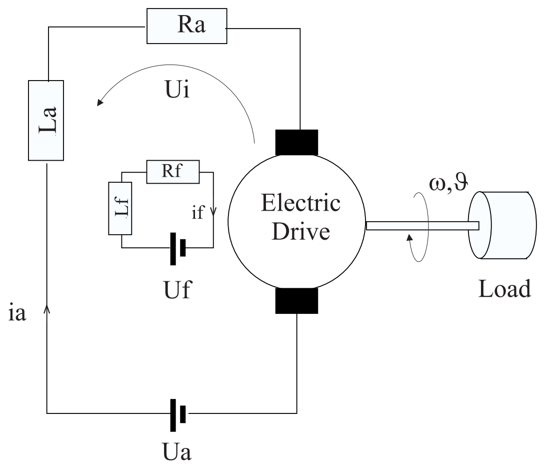


Figure 2.4. Electro-mechanical device

mechanics and electric circuits. The procedure of structured modeling usually means that a system is first divided into components down to a level where each component is simple enough (considering the electric-drive model as an elemental component will suffice for the section objectives). This recursive decomposition is supported to different extent by the modeling and simulation tools discussed in this section.

Magnetic fields is one of the possible ways of interaction between electrical and mechanical systems. This is how the electric drive, whose diagram is shown in Figure 2.4, works. The device has two separate coils, the *armature* coil, which is mounted on the rotating part of the device, and the *filed* coil, which is mounted on the stationary part of the device. The current flowing through the field coil generates a magnetic field. If current flows through the armature coil as well, a force responsible for the rotation is generated. The electric drive is said to be a *DC motor*. But if a torque is applied to the drive shaft, a current is generated in the armature coil. In this case, the electric drive is said to be a *generator*. Thus, two types of behaviour may be observed depending on the actual *"input"* excitation, establishing the *cause-effect* physical relationships. Independently on which one acts as cause or effect, the mechanical torque and the coil currents are related by the following expression:

$$\tau_m \;=\; k * i_f * i_a \tag{2.1}$$

Let us consider that, as it often happens, the field coil current i_f is kept constant. Thus, there is a linear relationship between torque τ_m and armature current i_a:

$$\tau_m \;=\; \psi * i_a \tag{2.2}$$

Operating as a DC motor, such configuration is called *armature control.* In this case, the constant ψ is called *torque constant.* Sometimes it is also called *Back EMF constant* since it also appears in the expression

$$u_i \;=\; \psi * w_m \tag{2.3}$$

which relates the voltage induced in the armature coil under the rotation influence (rotation velocity w_m). The equations 2.2 and 2.3 describe the coupling between the mechanical and the electrical part of the electric drive. The representation of the electrical and mechanical subsystems remains to complete the model. The armature and field circuits are represented by the mesh equations:

$$\begin{aligned} u_a \;-\; u_i &= R_a * i_a \;+\; L_a * \frac{di_a}{dt} \\ u_f &= R_f * i_f \;+\; L_f * \frac{di_f}{dt} \end{aligned} \tag{2.4}$$

Usually, the inductances L_a and L_f are small enough and their influence on the drive dynamics may be neglected. Thus, the electric part model can be simplified to the expressions:

$$\begin{aligned} u_a \;-\; u_i &= R_a * i_a \\ u_f &= R_f * i_f \end{aligned} \tag{2.5}$$

Finally, the mechanical part can be represented from the Newton's second law for rotational motion:

$$J_m * \frac{d^2\theta}{dt^2} = \tau_m - \tau_L - B_m * \frac{d\theta}{dt} \tag{2.6}$$

where J_m represents the drive mass inertia, B_m represents the inner frictions and τ_L represents an external mechanical load. The complete model for the electrical drive is shown in

$$\begin{aligned}
u_a - u_i &= R_a * i_a + L_a * \tfrac{di_a}{dt} \\
u_f &= R_f * i_f + L_f * \tfrac{di_f}{dt} \\
J_m * \tfrac{dw_m}{dt} &= \tau_m - \tau_L - B_m * w_m \\
w_m &= \tfrac{d\theta}{dt} \\
\psi &= k * i_f \\
\tau_m &= \psi * i_a \\
u_i &= \psi * w_m
\end{aligned} \tag{2.7}$$

Looking at Figure 2.4 for a more clear observation, three different *inputs* may be considered. These are: the voltage applied to the field coil u_f, the voltage in the armature circuit u_a and the torque load τ_L. The derived model represents, after some assumptions and simplifications, the phenomena that occur when one of these inputs is excited.

The ODE system in Equation 2.7 is the formalism that has been used to represent this physical behaviour derived from the first principles of electric and mechanics at the non causal specification level. Even so, the structure of the equations in 2.7 is not suitable for a numerical solver (it does not fulfill the *Simulator relation*). Hence, it must be manipulated in order to build *numerically solvable* models (see for instance Equation 2.9). The manipulation of such a behaviour formulation basically consist in a symbolic processing of the equations in order to find the proper computational structure.

We have classified the modeling methodologies into two families: the *causal* and the *behavioural* modeling tools according to the requirements they pose to the computational structure of the mathematical model. The degree of automation of the manipulation process goes from the total manually process (causal tools) to the *quasi* automatic computational causality assignment procedure (behavioural tools).

2.2.1. Causal modeling tools

The mathematical representation of a physical system can be usually expressed as sets of ordinary or partial differential equations. A common approach is to look at such dynamical system models as *black boxes* where the inputs and the outputs must be defined explicitly . This has been the approach traditionally adopted in, for instance, control theory, where dynamical systems are represented as transfer functions or state space realizations (Kailath 1980). From this point of view systems are as *processors*: they are influenced through its inputs, acting as causes, producing the outputs, the effects, together with the initial conditions and the system dynamics. This is a common trend in the following causal modeling tools since all of them require a mathematical model where the inputs, and consequently the outputs, are predefined.

Continuous system simulation tools

Despite having the simulation label, these tools may be considered as modeling environments since they make a slight separation between the behaviour representation and the simulation model by introducing certain modeling facilities. They are also named as *equation-oriented* modeling tools in the literature. Surveys on continuous simulation tools can be found in (Kreutzer 1986, Kheir 1986) or in (Matko *et al.* 1992). The main characteristic of this modeling framework is the formalism used to represent dynamic system behaviour. While differing basically in terms of syntactical details, the continuous system simulation tools are based on a *state-space representation* of the system, i.e., on a set of first-order ordinary differential equations (ODE) of the form:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, t) \end{aligned} \tag{2.8}$$

where $\mathbf{x}$ is the system state vector, $\mathbf{u}$ is the predefined input vector and $\mathbf{y}$ is the predefined output vector. There are many ways to find approximate numerical solutions to this ODE system. The methods are based on the idea of replacing the differential equation by a difference equation. These methods (e.g. Euler, Runge-Kutta or multi-step methods (Gustafsson 1992)) were very well known when digital simulators emerged in the 1960s. Most of the available general-purpose continuous time simulation tools are based on CSSL standard (Strauss 1967). The CSSL was a major milestone since it unified the concepts and language structures of the available simulation programs. A survey on CSSL tools is given in (Rimvall and Cellier 1986) and its use is well illustrated in (Cellier 1991).

A model formulated in terms of ODE systems as Equation 2.8 gives a causal explanation of the system behaviour, once the cause-effect physical relationships have been specified. In other words, there is a direct mapping between the physical system context and the formulation of the equation system. The structure of this formulation establishes a computational sequence where every variable can be solved from previously computed variables. For instance, in order to achieve a suitable ODE formulation for the mathematical model in 2.7, it is necessary to decide first whether the electric drive system will work as a DC motor or as a generator. This task sets the system input (physical causes) and the observable system outputs (physical effects). Then it is a question of finding the adequate computational structure reflecting these physical relationships. In the DC motor device, the causes are the voltages applied to the coils and the effect is the generated torque. This is reflected by the following ordinary differential system of equations:

$$\begin{aligned}
i_a &= u_a/R_a \\
i_f &= u_f/R_f \\
d\theta/dt &= \omega \\
d\omega/dt &= (\tau_m \; - \; \tau_L \; - \; B_m * \omega)/Jm \\
\psi &= k * i_f \\
\tau_m &= \psi * i_a \\
u_i &= \psi * w_m
\end{aligned} \tag{2.9}$$

For simplicity reasons, the coil inductances have been considered small enough to ignore their influence on the motor dynamics.

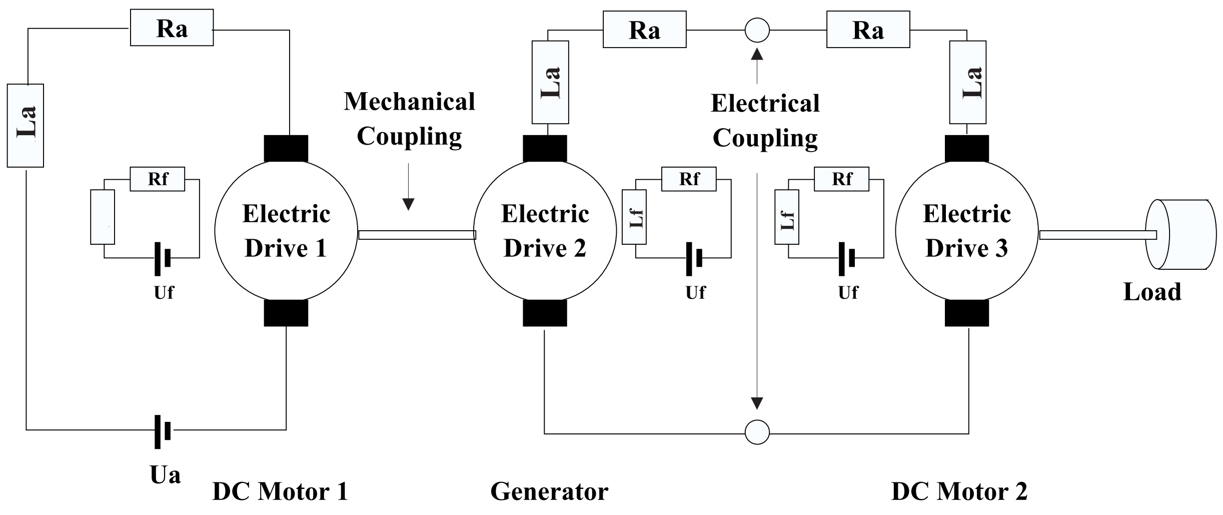


Figure 2.5. Ward-Leonard group.

Compared with the model formulation in Equation 2.7, the ODE formulation in 2.9 shows the computational structure suitable for the physical causality established in the DC motor operational mode. It should be noted here that this model formulation is assuming a particular context for the use of the electric drive. An important consequence is that this model can only be used in those cases where the electric drive operates as a DC motor. For the generator operational mode the physical context varies and the proper ODE model formulation should be determined. This a serious constraint to allow the reuse of ODE models since different reusing contexts for the same system lead to different simulation models.

The approximation to system model structuring and component model reuse in the CSSL simulation tools is the *macro* facility. The user may define models for system components and code them as macros which can be reused as components in a larger model. The macro is coded once and it is automatically expanded every time the component appears in a larger model. Consider the *Ward-Leonard* group shown in Figure 2.5 as a structured system composed by the coupling of three electric drives. This is an example where the same physical components show different cause-effect physical relationship (DC motor and generator). In order to show the lim-

```
MACRO DCMotor(theta,Omega,ua,uf,...!<=
 taum,tauL,Ra,Rf,Jm,Bm,JL,Omega0,theta0)!<=
 MACRO REDEFINE if,ia,ui,psi !<=
 MACRO REDEFINE derOmega !<=
  if = uf/Rf
  ia = (ua-ui)/Ra !<=
  derOmega=(taum-tauL-Bm*Omega)/(Jm+JL) !<=
  Omega = integ(derOmega,Omega0)
  theta = integ(Omega,theta0)
  psi = k*if
  taum = psi*ia
  ui = psi*Omega
MACRO END
```

Listing 2.1. Macro definition of the DC motor model

```
MACRO DCGenerator(theta,Omega,ia,uf,...!<=
 tauIn,Ra,Rf,Jm,Bm,Omega0,theta0)!<=
 MACRO REDEFINE if,ui,psi !<=
 MACRO REDEFINE derOmega,taum !<=
  if = uf/Rf
  ia = ui/Ra !<=
  derOmega = (tauIn-taum-Bm*Omega)/Jm !<=
  Omega = integ(derOmega,Omega0)
  theta = integ(Omega,theta0)
  psi = k*if
  taum = psi*ia
  ui = psi*Omega
MACRO END
```

Listing 2.2. Macro definition of the generator model

itations of reusing models which are formulated as ODE systems the electric drive CSSL model will be considered. Since CSSL tools require an explicit ODE formulation, the mathematical model in Equation 2.7 should be manipulated both to represent the DC motor behaviour of the electric drives 1 and 3 in Figure 2.5 (as the ODE system in Equation 2.9 does), and to represent the generator behaviour of the electric drive 2.

Listing 2.1 shows the macro for the DC motor behaviour and Listing 2.2 shows the macro for the generator behaviour. Significant differences may be observed in the code lines marked with `!<=` and in the input-output relationships represented by the macro parameterizations. In the motor macro, the equations establish that the voltages `ua` and `uf` and the torque load `tauL` are the causes (inputs), while the generated torque `taum` is the effect (output). This is just the opposite in the generator macro, where the equations establish that the the applied torque `tauIn` is the cause (input), while armature induced current `ia` is the effect (output). These differences appear because the representation formalism in the CSSL tools must reflect the physical causality relationships. As it is shown in the Ward-Leonard group example, the causal relationships can not be determined uniquely by analyzing a component, since they usually depend on the context where the component is used. The consequence of specifying the system behaviour at the causal explanation level is that different modeling components (macros in this approach) must be defined for the same physical component.

The ODE formulation as representation formalism does not contain any explicit information about the modeled physical behaviour. It is just a formalism very useful to apply to the computational mechanisms used to solve the model but, according to Definition 1.7, is a static formalism. The main problem of such formalism is that the modeling tool has no means to offer clever hints in order to help to the macro user to choose the proper macro for a given reusing context. Therefore, the macro user has to analyze the component reusing context and decide which macro suits in each situation. This may seem obvious in some cases as for instance the electric drive example, where the reusing context offers a quite clear decision on which role the component is playing. However, a much more deep analysis will be usually required. For instance, deciding if a simple electric resistor must be represented by the Ohm's law solved for the current ($I = V/R$) or for the voltage drop ($V = I * R$) is only possible after the complete analysis of the whole electric circuit.

Many other examples, where the same physical component has one ODE explicit formulation for each different reusing context, can be found in different application domains. Since the user has to analyze which formulation suits better in the reusing context, we can conclude that with macros the only modeling *"work saving"* is the tedious task of re-writing the same piece of code in different parts of the model. This is a quite error safe facility but gives a poor measurement of the CSSL tools automated modeling level. From this point of view, the reusing facility requires a representation formalism where the physical causality does not have to be predefined in terms of an explicit mathematical formulation.

Block-oriented graphical modeling

Block-oriented approaches mainly address modeling on a graphical system representation. The system model is defined by a diagram consisting of the connection of blocks where the behaviour of a system component or a part of it is modelled. These blocks are standardized in the sense that they define an input-output relationship. A block may represent a simple mathematical operation (as a sum or an integration), a function, a transfer function, linear and non linear systems in state-space form, etc. These model blocks are usually organized in libraries. New models representing some input-output physical relationship can be defined by connecting predefined blocks. These blocks are linked by *signal-like* connections representing the flow of information. Different graphical modeling tools appeared when computers with graphical facilities became widely available. The *Simulink* (Mathworks 2000*b*) modeling toolbox for MATLAB (Mathworks 2000*a*) or the graphical interface to ACSL, *Graphics Modeller*, are examples of such tools.

To illustrate the block-oriented approach, the Simulink model of the Ward-Leonard group shown in Figure 2.5 will be defined. The model is structured in blocks representing the two main system components: the DC motor and the generator.

Since a block represents an input-output relationship, the same constraints described above for the ODE formulation are found when the reusing context of a system component establishes different cause-effect physical relationships. Therefore, different modeling blocks should be defined to represent both the electric drive DC motor and generator operational modes. Figures 2.6 and 2.7 show the Simulink blocks representing the DC motor and the generator behaviour respectively. As it can be observed, the computational structure defined in these models is analogous to the computational structure of the ODE systems represented by the macros in Listings 2.1 and 2.2. Actually, there is no essential difference between the plain ODE formulation and its graphical representation in terms of arithmetic operator connections. This representation is rather similar to the system representation used in the early days of analog simulation, although present graphical modeling tools have building blocks more sophisticated (e.g. *s-functions* of Simulink), which anyway predefine some computational causality.

The models in Figures 2.6 and 2.7 can be masked as basic blocks, with ports describing the input-output interactions, in order to be incorporated in a model library and be reused as submodels in new model constructions (see for instance Figure 2.9). However, several considerations have been taken into account before the block definition, attending to their future role as a part of a larger model. Let's consider first the coupling between the DCmotor 1 and the generator. The torque produced by the motor is transmitted to the generator in order to produce an electrical signal. With this connection, the mechanical link makes the two angular velocities to be the same, except for the sign because of symmetry reasons:

$$\omega_1 \;=\; -\omega_2 \tag{2.10}$$

This physical constraint forces to consider once again the mathematical formulation of the system, since the addition of Equation 2.10 as the effect of the mechanical coupling introduces a significant change in the model structure. Now, the motion dynamic is described by the system:

$$\begin{aligned} J_{m1} * \frac{d\omega_1}{dt} &= \tau_{m1} \;-\; \tau_{L1} \;-\; B_{m1} * \omega_1 \\ J_{m2} * \frac{d\omega_2}{dt} &= \tau_{m2} \;-\; \tau_{L2} \;-\; B_{m2} * \omega_2 \\ \omega_1 &= -\omega_2 \end{aligned} \tag{2.11}$$

This *Differential-Algebraic* system of equations (DAE) requires of specific numerical solvers, such as DSSL (Brenan *et al.* 1989), which are not presently provided by most of the block-oriented modeling tools. Hence, some manipulations have to be done in order to avoid this algebraic problem once the motor and generator blocks are interconnected. The DAE system in Equation 2.11 is said to be of index one (see for instance (Brenan *et al.* 1989) for a discussion on DAE index) and must be reduced to zero index system in order to apply and ODE solver. This reduction will be done here by considering the physical interactions resulting from the mechanical coupling (a generic method for index reduction can be found in (Pantelides 1988*a*)).

Due to the mechanical coupling, the torque produced by the DC motor represents the torque load of the generator, and the torque produced with the generator rotation is the torque load for the DC motor. The same reasoning applies to the inertia, since the inertia of each drive constitutes the inertial load of the other drive. So the motor-generator motion dynamics can be

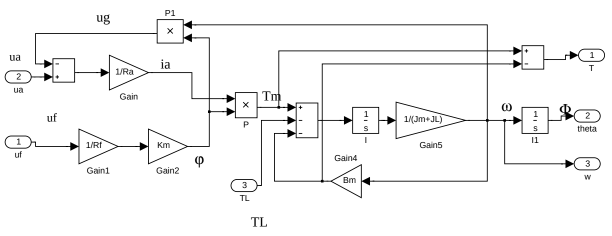


Figure 2.6. DC motor Simulink model

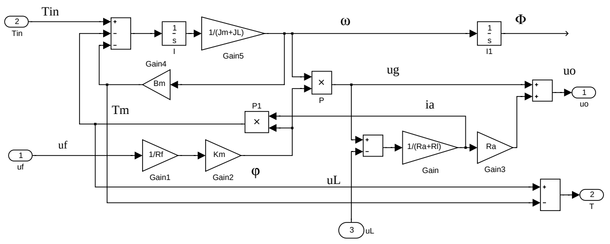


Figure 2.7. Generator Simulink model

written as:

$$
\begin{aligned}
(J_{m1} + J_{L1}) * \frac{d\omega_1}{dt} &= \tau_{m1} \; - \; \tau_{L1} \; - \; B_{m1} * \omega_1 \\
(J_{m2} + J_{L2}) * \frac{d\omega_2}{dt} &= \tau_{m2} \; - \; \tau_{L2} \; - \; B_{m2} * \omega_2
\end{aligned}
\tag{2.12}
$$

the coupling equations:

$$
\begin{aligned}
\tau_{L1} &= \tau_{m2} \; - \; B_{m2} * \omega_2 \\
\tau_{L2} &= \tau_{m1} \; - \; B_{m1} * \omega_1
\end{aligned}
\tag{2.13}
$$

and the parameter assignment:

$$
\begin{aligned}
J_{L1} &= J_{m2} \\
J_{L2} &= J_{m1}
\end{aligned}
\tag{2.14}
$$

A straightforward analysis of these equations concludes that $\omega_1 = -\omega_2$, so this equation does not have to be included and the DAE system is avoided. The price to be paid is the waste of computing time since the same angular velocity is calculated twice as the solution of the system in Equation 2.12, but the mechanical coupling constraint in Equation 2.11 is guaranteed. The block diagrams of this model, where the DAE is avoided, can be observed in Figures 2.6 and 2.7.

Some algebraic questions also emerge in the electrical coupling between the generator and the second DC motor of the Ward-Leonard group. Figure 2.8 shows this link. In order to have modeling blocks where components are represented separately, the electrical behaviour of each mesh can be expressed as:

$$u_o = R_g i_g + u_g \tag{2.15}$$

$$u_a = R_a i_a + u_i \tag{2.16}$$

and the coupling equation

$$\begin{aligned} u_o &= u_a \\ i_g &= -i_a \end{aligned} \tag{2.17}$$

A modeling block computes the output from the block inputs so, as the equations in ODE explicit formulations, Equations 2.15, 2.16 and 2.17 must be arranged in order to suit the computational structure:

$$\begin{aligned} i_g &= (u_o - u_g)/R_g \\ u_o &= u_a \\ u_a &= u_i + R_a i_a \\ &= u_i - R_a i_g \end{aligned} \tag{2.18}$$

These equations can be separately included in the generator and the DC motor blocks by considering the voltage u_a as the electric load in the generator, and the current i_g as the actual input for DC motor block. However, the system in Equation 2.18 constitutes a so called *algebraic loop*, since there is a cyclic dependence involving the variables i_g, u_o and u_a. An algebraic loop requires of explicit solvers in order to break (tear) numerically the cyclic dependence. This is always an inefficient solution and sometimes may even lead to unstable numerical situations if the tearing variable or its initial value is not properly selected. This problem can be avoided if a different output-input relationship between the generator and the motor is considered. For

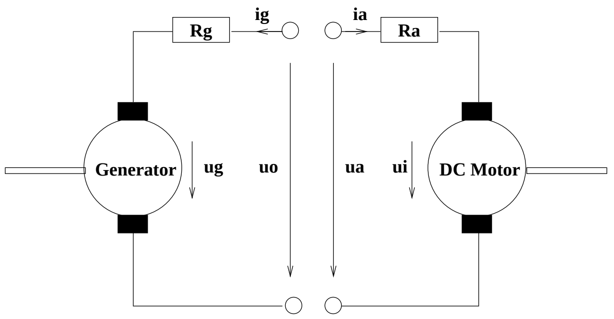


Figure 2.8. Electrical coupling between a generator and a DC motor.

instance, the DC motor may be treated as resistive load in the generator and the motor induced voltage can be included as the resistance load:

$$\begin{aligned} i_g &= (u_L - u_g)/(R_g + R_a) \\ u_o &= u_g + R_g i_g \end{aligned} \tag{2.19}$$

This solution avoids the algebraic loop in the generator-motor coupling and has been implemented in the models at Figures 2.6 and 2.7. On the one hand, this model overcomes the algebraic problem; on the second hand, there is an *a priori* assumption in the generator modeling block, which is including information about the parameters and the behaviour of a modeling block which only potentially is going to be connected to it. Therefore, this *ad-hoc* modeling solution will not work if a component with a different electric load is connected to the generator.

Finally, the aggregated simulink model of the Ward-Leonard group is shown in Figure 2.9. As it can be observed, the block connections respond more to the algebraic constraints discussed previously than to the physical aspects of component interactions. Careful attention should be paid to the mathematical structure of aggregated models in order to avoid the undesirable consequences of coupling modules with explicit mathematical formulations (i.e., high index DAE systems or algebraic loops). This is directly extensible to the macro issue in the CSSL languages. Macros, as blocks do, predefine a computational input-output relationship.

Except for providing with a *friendly* graphical model representation instead of the more te-

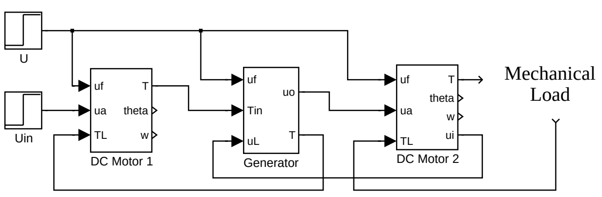


Figure 2.9. Ward-Leonard group Simulink model.

dious textual representation of the CSSL tools, there is no significant improvement in the level of automated modeling. The modeller still has to obtain the proper mathematical structure of the model according to the system physical reusing context. The topology of block connections represents this mathematical structure but, in general, it does not match the topology of the represented system. Furthermore, the modeller must carefully analyze the mathematical structure of the aggregated models and find the solutions to the possible algebraic problems derived from the modeling block coupling. In spite of the modular appeal of this modeling approach, blocks (as well as macros) are not real reusable modular structures since the physical context (cause-effect relationships) must be predefined in order to formulate the model. Hence, the model formulation must usually include assumptions on the component models to which is going to be connected.

Flowsheeting in Process Engineering

Process simulators, often termed as flowsheet simulators, are tools which are used to build models of manufacturing facilities used for processing and transforming materials. The flowsheeting methodology can be considered as domain-specific since it focus on process engineering. Obviously, we can not address the discussion through the Ward-Leonard example. A short discussion on this approach is introduced here because some of its methodological modeling aspects to structure the knowledge about the system have been incorporated to the design of the PML physical knowledge representation structures.

The process model description is basically separated into three parts: the process topology, the process units and the processed medium or material. Every process is abstracted by a block diagram consisting of standardized blocks modeling the behaviour of a process unit or a part of it (they can be subsumed as modeling components according to Definition 1.3). Process unit models may represent either the steady-state or dynamic behaviour of a process. The processed medium is usually described in terms of the so called process quantities (also medium properties). The blocks are linked by connections representing the stream flows of information, material or energy. This modeling approach offers an appealing way to build models since blocks are connected according to the system flowsheet topology.

Two different approaches or architectures to model representation and simulation are distinguished in flowsheeting simulators (Westerberg and Benjamin 1983, Marquardt 1991): the *procedural* or sequential modular and the *declarative* or equation-oriented.

In the procedural approach the representation of a unit model contains the behaviour equations as well as the method to solve them. No restrictions on either the equation representation formalism or on the solution algorithm are imposed as long as the input and the outputs are predefined according to the specified stream standard. Hence, the procedural approach has many similarities with the block-oriented approach, even here the blocks are predefined as process unit models as well as the streams as their input/output relationships. Thus, the only issue expected from a block is to produce the outputs given the inputs produced by the predecessor block. When numerical loops appear due to recycles, they iterate on the sequence until it converges. Many commercial tools are based on this approach (references to these tools can be found in (Westerberg and Benjamin 1983, Marquardt 1991, Marquardt 1996)). That is the reason for including them in the causal modeling tool section. According to the specification levels described in Chapter 1 (see Table 1.1), the system is specified at the causal explanation level despite it may seem that it is specified at the topological level. Therefore, many of the reusing limitations shown by block oriented or CSSL tools are also present in the procedural approach due to the predefined reusing context of the model building blocks.

However, a different consideration should be made with some the declarative or equation-oriented flowsheeting simulators. In the declarative approach, the procedural knowledge (solution algorithms) has been segregated from the declarative knowledge (behaviour equations).

Some languages, such as SPEEDUP (Pantelides 1988*b*, Pantelides and Barton 1993), are closer to the behavioural approach discussed in the section bellow.

2.2.2. Behavioural modeling tools

Looking to models as *input/output processors* leads to behaviour representations which are difficult to reuse and whose formulation requires of manual manipulation of the equations. An obvious limitation in this approach is the input/output causality predefined in the representation formalism. Nowadays there is a growing interest in many engineering disciplines for having a more generic definition for dynamical system models, without an *a priori* assumption on the cause-effect relationships. Such a generic definition can be found in (Willems 1991):

Definition 2.1 – *Dynamical System*

A *dynamical system* Σ is a triple $\Sigma = (\mathbf{T}, \mathbf{W}, \mathcal{B})$ with $\mathbf{T}$ the *time axis*, $\mathbf{W}$ the *signal space*, and $\mathcal{B} \in \mathbf{W}$ the *behaviour*. ❑

The sets $\mathbf{T}$ and $\mathbf{W}$ define the mathematical aspects of the model: the time set and the space in which the time signals produced by the system take on their values. $\mathcal{B}$ formalizes the laws which govern the system.

The aforementioned definition can be easily extended to the *interconnection of Dynamical systems* (Willems 1991). A interconnected system can be defined as $\Sigma = \Pi_\alpha \Sigma_\alpha = (\mathbf{T}, \mathbf{W}, \mathcal{B})$, where Σ_α denotes the set of system components and Π_α represents their connection topology. Let $\Sigma_1 = (\mathbf{T}, \mathbf{W}, \mathcal{B}_1)$ and $\Sigma_2 = (\mathbf{T}, \mathbf{W}, \mathcal{B}_2)$, the interconnection of Σ_1 and Σ_2 is defined as $\Sigma_1 \wedge \Sigma_2 = (\mathbf{T}, \mathbf{W}, \mathcal{B}_1 \Cap \mathcal{B}_2)$.

The Definition 2.1 can be considered the starting point of the behavioural approach as a foundation framework for the theory of dynamical systems (Willems 1991). An application to control theory can be found in (Willems 1997).

For our modeling purposes, this definition matches with our understanding of the model as a way of capturing the notion of system behaviour. In this context, there are several aspects in Definition 2.1 and in its extension to interconnected systems which should be commented: it does not make any reference to input/output relationships and it does not impose the formalism to describe the behaviour $\mathcal{B}$ (e.g. references are not made nor to variables and nor to equations).

There are several reasons for not defining in a model an *a priori* distinction between the inputs and the outputs. First, the input/output representation can be deduced from such a model as an special case when the model application is defined (its reusing context is stated). For example, a behavioural model of a electrical resistor may represent the Ohm's law and, according to the boundary conditions imposed by the physical context, the model can be mathematically formulated either as $V = IR$ or $I = V/R$. Second, which variable will be used as input or output in the model of a physical device may depend on the purpose for which it will be used. For example, will the electro-mechanical device model in Figure 2.4 be used as a DC motor or as a current generator?. We claim that this question can only be answered once the model reusing context has been stated.

Several modeling frameworks currently available fit with the Definition 2.1, often called as *behavioural system* since it focus on the behaviour aspects rather than on the equations that represent the system. In most of the modeling languages $\mathcal{B}$ is described by means of behavioural equations: sets of equations without a predefined computational causality (i.e. the equals operator describes a equality relationship between math expressions and not an assignment operation) and whose solution describes the system behaviour. PML makes a different interpretation about how $\mathcal{B}$ should be represented. The PML modeling language does not focus on the behavioural equations as the means to express the physical behaviour. Instead, $\mathcal{B}$ representation emphasizes the physical aspects of the system and delays its mathematical formulation to the experimentation purposes.

Either using behavioural equations or using a different formalism (e.g. PML) to represent the behaviour, the modeling languages developed under the object oriented paradigm are considered *behavioural modeling tools* (Andersson 1990). This is the case of the object-oriented modeling tools such as Dymola or EcosimPro. An analogous interpretation of the behavioural approach is made by the *Bond Graph* modeling methodology (Thoma 1990, Cellier 1991). For instance, TwentySim (Controllab Products B.V.,1999) based on CAMAS (Broenink 1990) is a modeling tool designed under this approach.

These modeling methodologies put the emphasis in describing the structure (topological or phenomenological) of the system by connecting modeling components which capture the essential behaviour of the system components. These modeling components do not make any assumption on the input-output causality since it can be deduced from the declared behaviour according to their reusing context.

The way in that the aggregated behaviour $\mathcal{B}_1 \Cap \mathcal{B}_2$ of two interconnected systems $\Sigma_1 \wedge \Sigma_2$ is deduced from the model depends on the formalism used to represent $\mathcal{B}$. The aggregated behaviour $\mathcal{B}_1 \Cap \mathcal{B}_2$ should be understood as the intersection of behaviours as a way of formalizing that additional laws are imposed on the interconnected system. A major discussion in this thesis, and a contribution of the PML language, is to guarantee that the aggregated behaviour deduced from interconnected models does really represent the aggregated behaviour of the corresponding interconnected systems. Chapter 3 is mainly devoted to this question.

The two sections bellow describe which are the mechanisms used by the behavioural modeling tools based on equations in order to obtain the aggregated behaviour $\mathcal{B}_1 \Cap \mathcal{B}_2$ of two interconnected models.

Equation-based Object Oriented modeling

The behavioural approach is motivated by the way physical systems are conceptualized and modeled from physic principles. Analogous ideas have motivated the object-oriented modeling framework which has been adopted in this thesis and whose basic principles can be summarized as follows:

- The modeling components encapsulate the representation of the physical behaviour $\mathcal{B}$ by explicitly defining an interaction mechanism (see Definition 1.3). Hence, what is important to reuse a modeling component is the interaction representation (the accessible part of the modeling component), and not the internal representation.

- Systems are specified at a structure level as interconnected modeling components ($\Sigma = \Pi_\alpha \Sigma_\alpha$). This structured approach can be recursively applied to define new modeling components with ever-growing complexity.

- In order to make the connection structure Π_α independent from the behaviour $\mathcal{B}_\alpha$ represented in Σ_α, there is not a causal *a priori* assignment of the inputs and outputs in the interaction representation of the modeling component. The aggregated behaviour is derived from Π_α together with the behaviour $\mathcal{B}_\alpha$ encapsulated in Σ_α.

A deeper analysis of the methodological aspects of the object orientation is left to Section 2.3. Here we address the discussion towards those environments whose behaviour representation formalism is based on equations. We qualify or label as equation based to those languages where behavioural equations are used to represent the physical behaviour. These equations usually have an appealing form giving a convincing interpretation of such things as balance equations, conservation laws, constitutive laws, etc. However, it should be remarked that, whereas the equation uniquely describes such type of physical behaviour, the contrary is not true. That is, there is not an unique possible math formulation of the physical behaviour. This is what we have qualified as implicit representation of physical behaviour in Chapter 1. The main consequence we want to show is that such formalism should be considered as *static* according to Definition 1.7. As we will discuss through this thesis (see Chapter 3), this fact constrains seriously the reusability of the mathematical modeling components.

The behaviour equation introduces the concept of variable as the representation of some physical quantity usually time dependent (in a discrete or continuous way). In the object oriented approach, the variables are separated into internal and external (Andersson 1994).

Since the behavioural approach focus on the behaviour manifested by the external variables, the internal variables are usually introduced for convenience, even they appear in a natural way when the behaviour is declared from the physics first principles. For instance, the state variables and their derivatives are an example of internal variables in a model representing the system dynamics. The internal and external variables can be assimilated to the *latent* and *manifest* variables in the behavioural framework defined in (Willems 1991).

We do no pretend to give here an exhaustive description object-oriented approach adopted by several modeling tools based on equations, such as Omola, Dymola (based on the Modelica language) and EcosimPro. Fundamental readings about this matter can be found in (Cellier 1991, Andersson 1990, Nilsson 1993, Andersson 1994, Elmqvist et Al. 2000, Cobas and Al. 1999).

We just try to describe how the behaviour is encapsulated in the modeling components by means of a mathematical interface. In the object oriented approach the modeling components are based on the class representation structure (see (Meyer 1997) for a thorough description of the class concept in object orientation). The modeling class encapsulates the representation of the system behaviour. The visible part of the modeling class is defined by the parameters (data used by the user to characterize a particular instance of the class) and by the coupling mechanism (interface defined by the class in order to interact with other modeling classes). In the equation based tools, the coupling mechanism is defined by a mathematical interface representing the variables shared between the modeling component and its environment. The variables of the modeling class interface are classified into *across* and *through* variables. The distinction between both types gives the mechanism to represent the physical interactions among two coupled models. The following equations are generated when two, or more, models are coupled through this interface:

Across. The connection of n across variables generate an equality equation such as:

$$e_1 = e_2 = ... = e_n \tag{2.20}$$

Through. The connection of n across variables generate an sum to zero equation such as:

$$\sum_{i=1}^{n} f_i = 0 \tag{2.21}$$

The set of coupling equations derived from the model connection structure (Π_α), together with the behaviour equations ($\mathcal{B}_\alpha$) encapsulated at each modeling class is used to derive the aggregated behaviour. The obtained set of equations are manipulated in order to get a formulation suitable for the numerical solvers (fulfillment of the simulator relation). This translation procedure, defined as computational analysis procedure (see Definition 1.6), moves a system specification made at a structure level into the causal explanation level (see Table 1.1). An explanation of this procedure can be found in (Cellier and Elmqvist 1993).

Example 2.1 – *A modeling class for a resistor using the Modelica language*
The model for an electrical resistor is developed in order to illustrate how does it work the mathematical interface of a modeling class (the Modelica language is used for this purpose).

The resistor behaviour can be represented by the Ohm's law $V = IR$, where R is the electrical resistance, V is the voltage drop across the resistor and I is the throughput current. In order to compute this mathematical relation, the voltage at both resistor poles and the current through them should be known. This information, which defines the interaction with other electrical components, is represented in the equation based language by means of the interface variables. The Modelica declaration for this interface could be:

```
connector electricalTerminal
  Voltage v; % across variable
  flow Current i; % through variable (flow qualifier)
end electricalTerminal
```

The v and i *manifest* variables define which of the *latent* variables can be shared among coupled modeling classes. We can now define the modeling class representing the resistor behaviour. The Modelica declaration for this class could be:

```
model Resistor
  electricalTerminal T1,T2
  parameters Real R;
  equation
     V=T1.i*R; % Ohm's law
     V=T1.v-T2.v; % Potential drop
     T1.i+T2.i=0; % Current through resistor
```

```
end Resistor
```

The dot notation (e.g. `T1.i`) is used to relate the behaviour equations with the model class manifest variables (the model interface). When we connect this resistor model with other electrical component models through the `electricalTerminal` interface, two equations are generated: an equality among the voltage variables at the connection node (see Equation 2.20) and a sum to zero of the current variables (see Equation 2.21). Actually, the coupling equations are representing the Kirchhoff's law. ❑

When the behaviour representation is based on equations the modeller has to predefine which phenomena are represented by the modeling class (see Example 1.4). Hence, the modeling class is predefining in which phenomenological structure can take part (the phenomena of interest have been set). The behavioural equations are also predefining the model adequacy (the equation states the accuracy used to formulate the behaviour).

As Example 2.1 illustrates (see also Example 3.1), the model class mathematical interface is constituted by variables shared between the behavioural equations encapsulated by the model class and its environment. Hence, the modeling class interface depends directly on the behaviour which has been represented by means of behavioural equations. The straightforward implication is that the reuse of a modeling class is constrained to a context where the mathematical interface is able to represent the physical interactions derived from the two coupled components.

According to the basis set in Chapter 1, we may conclude that the specification of a system by means of an equation based object-oriented modeling tool is made at the phenomenological structure level.

In relation with the main characteristics demanded to a modeling tool (see Section 1.2), the object oriented approach based on the mathematical formalism:

- Supports a structured modeling approach.
- Does not support the system specification at the topological level. Under certain circumstances it may exist an analogy between the model structure and the system topology.

- The modeling language is not a dynamic formalism (see Definition 1.8) because of:
  - The physical reusing context is predefined by the modeling classes.
  - It does not provide with explicit physical knowledge representation structures.

The lack of these characteristics fails to fulfill with the criteria established to measure the modeling automation level:

- In relation to model construction, systems can not be represented at the topological level by means of reusability of predefined modeling components. The reusing physical context is predefined by the modeling classes, so the specification should be made at the phenomenological structure level. Since the mathematical formalism does not give explicit information of the represented behaviour, the modeller has to analyze the behavioural equations before taking the decision on the validity of a predefined modeling component (modeling class) within his structured model.
- Attending to its usage, a ready-made model specified at the phenomenological level is suitable to give a causal explanation of the system behaviour considered within certain experimental framework. Therefore, it can not be adapted to different simulation purposes. Obviously, it makes not sense to speak about tools automating the model adaptation for different simulation purposes.

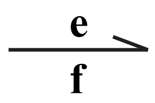
Bond-graph Modeling

The bond-graph approach (Karnopp and Rosenberg 1968, Thoma 1990, Cellier 1991) was originally conceived as a graphical tool to help engineers to represent the physical interactions present in a system in order to derive a causal explanation for the phenomena of interest. Nowadays we can find commercial tools where the system is modeled according to the bond-graph methodology (e.g. TwentySim Controllab Products B.V.,1999). These tools are able to manipulate this representation in order to set the behaviour causal formulation.

The bond-graph modeling methodology is a structured approach based on the description of the energy exchange between the components of a system. The system behaviour representation is structured in modeling components describing the phenomena of interest. The interaction between the modeling components is described in terms of energy exchange and it is represented in terms of power flow by means of two variables named *effort* (e) and *flow* (e), in such a way that the energy can be calculated as:

$$E = \int P dt = \int e \times f \; dt \tag{2.22}$$

The power flow is graphically described by means of a bond (harpoon indicates the flow direction):



The interface of the modeling components in bond-graph is named *port*. The behaviour encapsulated in a modeling component must represent the physical relation among the effort and flow variables specified at the ports by means of a non causal equation (behavioural equation):

$$e = \Phi(f) \tag{2.23}$$

where Φ is the mathematical formulation of the behaviour represented at the modeling component. The basic modeling components are usually classified in terms of the number of ports:

1-port basic modeling components. They represent constitutive relationships. They are classified into passive and active components depending on the declared phenomenon.

- *Resistive components.* It is a passive component used to represent energy dissipation. Is graphically represented with the symbol $\rightharpoonup R$. The behavioural equation has the form $e = \Phi_R(f)$.
- *Capacitance components.* It is a passive component used to represent energy accumulation. Is graphically represented with the symbol $\rightharpoonup C$. The behavioural equation has the form $e = \frac{1}{C}\int \Phi_C(f)dt$.

- *Inductive components.* It is a passive component used to represent energy accumulation. Is graphically represented with the symbol $\rightharpoonup I$. The behavioural equation has the form $f = \frac{1}{I}\int \Phi_I(e)dt$.
- *Sources.* They are active components used to represent energy addition into the system. They can represent an effort source (e.g. a voltage source) or a flow source (e.g. an electrical current source). They are graphically represented with the symbols $S_e \rightharpoonup$ and $S_f \rightharpoonup$. The behavioural equation has the form $e = \Phi_{S_e}(t)$ and $f = \Phi_{S_f}(t)$.

2-port basic modeling components. These components are used to represent the energy conservation principle in those physical systems where the feed energy is converted into another form of energy. For example, a DC motor can be viewed as a transducer where the electrical energy is transformed into mechanical energy. There are two types of transducer modeling components: *Transformers* and *Gyrators*. They represent an *ideal* energy conservation transducer governed by the equations:

$$\begin{array}{ccccccc} Transformer: & & & \quad & & Gyrator: & \\ e_1 & = & me_2 & & e_1 & = & rf_2 \\ f_1 & = & mf_2 & & e_2 & = & rf_1 \end{array}$$

where $e_1 \times f_1$ is the feed power flow and $e_2 \times f_2$ is the output power flow. They are graphically represented with the symbols:

$$\begin{array}{ccc} e_1 & & e_2 \\ \rightharpoonup & TF & \rightharpoonup \\ f_1 & & f_2 \end{array} \qquad \begin{array}{ccc} e_1 & & e_2 \\ \rightharpoonup & GY & \rightharpoonup \\ f_1 & & f_2 \end{array}$$

***multi*-port basic modeling components**. These modeling components have three or more ports. They behave as ideal components where no energy is lost, nor accumulated. The are called *junctions* since they are used to represent how the previous modeling components are connected. There are two types:

- *0-junction.* In this junction all the effort variables are equalized and the flow variables add up to zero.
- *1-junction.* In this junction all the flow variables are equalized and the effort variables add up to zero.

They are graphically represents by the symbols:

$$\begin{array}{ccc} \upharpoonleft & \qquad\qquad & \upharpoonleft \\ \rightharpoondown 0 \rightharpoondown & \qquad\qquad & \rightharpoondown 1 \rightharpoondown \end{array}$$

In difference with the object-oriented approach, where the behaviour represented in a modeling components (classes) and its interface is a design decision taken by the modeller, the modeling component in bond-graph are predefined. The modeller should determine how to formulate the phenomena represented at the 1-port modeling components, i.e., he should define the Φ equation according to the represented phenomena.

For example, let us consider the electrical circuits shown at Figure 2.10. In order to describe the energy exchange among the physical components, we may choose the electrical voltage as the effort variable and the current as flow variable. Their product gives the electrical power. The resistor behaviour can be represented by a *resistive* component R, the capacitor by a *capacitance* component C, the inductance by an *inductive* component I and the voltage source as an *effort source* component S_e. The remaining task is to determine how the energy flows must be represented according to the system topology. The electrical components are connected in series at the first circuit, so the current through them is the same and the voltage drop at each electrical component adds up to zero. This can be described by means of the 1-junction modeling component. In the second circuit, the electrical components are connected in parallel. Hence, the current through them adds up to zero and the voltage drop at each electrical component is equal. This can be described by means of the 0-junction modeling component. The bond-graph models of both circuits are shown at the right side of Figure 2.10. We may observe that, even the topology of both circuits is different, the model connection structure in both cases is the

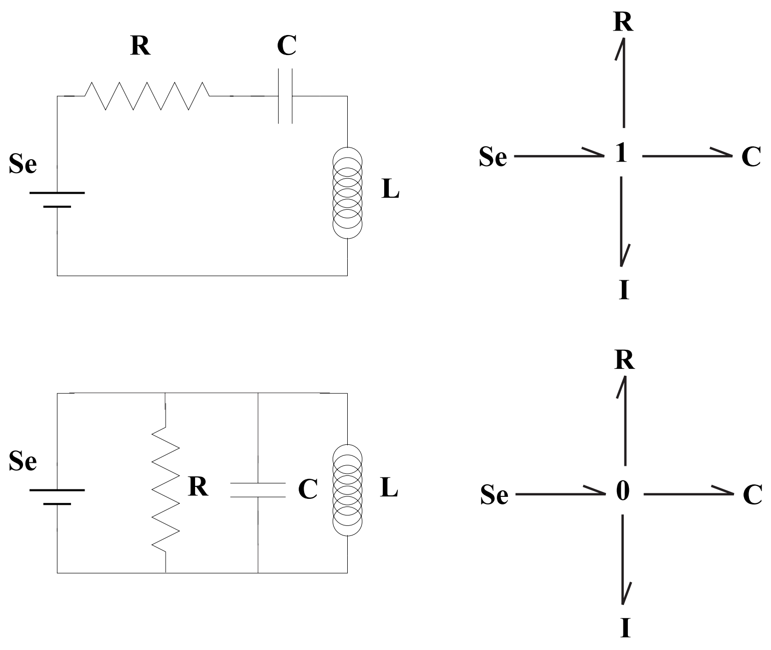


Figure 2.10. Bond-graph models of a RLC electrical circuit. The serial connected components illustrates the 1-junction and the parallel connected components illustrates the 0-junction.

same. The only difference is the junction modeling component. That is, the system topology has a direct influence on the modeling components (mainly the junctions) which must be selected to represent the aggregated behavior and not a so direct influence on the modeling components connection structure. This occurs because the bond-graph model is a system specification at the phenomenological structure level.

Once the modeller has decided which system phenomena is going to represent, i.e. sets the experimental framework, he has to derive the connection structure of the modeling components he needs to use, i.e., he sets the phenomenological structure of the specified system. The reused modeling components formulate the physical behaviour at the non causal explanation level, which implies that the model adequacy is already determined.

According to the modeling process described in Chapter 1, a bond-graph model is conceived at the phenomenological structure level. The modeling components are formulated at the non causal level. The connection mechanism (bonds) do not predefine the physical causality, they just represent a positive energy flow. The behaviour at the modeling components is expressed by means of non causal equations. Hence, the bond-graph should be translated in order to obtain a causal explanation. The *SCAP* procedure (Sequential Causality Assignment Procedure) is a wide spread algorithm (see for instance (Thoma 1990)).

2.2.3. Discussion

Modeling and simulation activities have been traditionally mixed up because simulation problems require causal model formulations (specification at the causal explanation level). This is more obvious in those tools which require a model formulation close to the numerical solver algorithms (e.g. causal modeling tools). Actually, in many of these approaches the technology (model numerical solution) has conditioned the modeling paradigm. A direct consequence is that model formulation is more related to the numerical solver aspects, so to say, the procedural knowledge (e.g. the causal modeling methodologies such as the classical equation or block oriented paradigms).

I would like to remark that this section had not pretended to be an intensive revision of the state of the art in the modeling domain. Very good classifications, descriptions and main characteristics analysis of most of the present commercial and academic modeling tools can be found in the literature. A short revision of the modeling tool evolution is given in (Åström *et al.* 1998), and a more intensive review on the state of the art as well as future trends in modeling and simulation in (Marquardt 1991) or (Marquardt 1996). We have tried to relate the main characteristics of the most extended approaches in order to analyze the modeling automation level achieved by them. These characteristics are summarized in Table 2.1. The discussed approaches present important differences in the methodology, the starting specification level and the application domain. However, it should be noticed that all of them use the mathematical equation formalism to represent both the physical behaviour (math equations constitute the language to express the first principles) and the numerical concerns (the mathematical formulation will be use to solve numerically the model).

Many of the characteristics shown by these modeling approaches are present in PML. How-

Modeling Method	Recursive Structured Approach	System Specification Level	Dynamic Formalism	Application Domain	Simulator Relation
Eq. Oriented	No	Causal	No	generic	Explicit
Block Or.	Yes	Causal	No	generic	Explicit
Flowsheeting	No	Causal	No	Ch. process	Explicit
Eq. OOM	Yes	Phen.	No	generic	CAP
Bond-Graph	Yes	Phen.	No	generic	SCAP

Table 2.1
Important factors to measure the automation degree achieved by different modeling approaches.

ever, a crucial difference is the representation formalism defined by the PML language. With PML, we introduce a clear separation between the modeling concerns and the simulation concerns. There are other modeling tools, as for instance Model.la (Stephanopoulos *et al.* 1990*a*, Stephanopoulos *et al.* 1990*b*), HPT (Woods 1993) or SIMPD (Acebes 1996), which propose representation formalisms where the behaviour representation and the simulator relation are treated as separated matters. One of the main reasons for this separation is to automate the modeling process.

The following discussion intends to give a measure of the automation achieved by the presented modeling approaches according to the criteria postulated at Section 2.1. The objective is to set a referential frame to locate the PML automation features according to the modeling process described in Chapter 1.

The model construction criterion

This criterion was formulated at Section 2.1 to state the main features which should be provide by a modeling tool in relation to model construction. Most important for a more sophisticated modeling process is to limit the involvement of users in the inclusion of existing a priori knowledge while making this process automated. Reusability of predefined modeling components has been shown as the mean to fulfill with this premise.

From the model construction perspective, the main limitations of the causal and behavioural equation approaches are a consequence of the representation formalism deficits: the mathematical formalism is a static modeling formalism (see Definition 1.7) and does not provide with explicit information about the represented physical behaviour. To lay the foundations of this assertion we may recall the boundaries of the mathematical formalism as the vehicle to represent physical behaviour and knowledge.

The causal mathematical formalism (e.g. explicit ODE, DAE or DE) used by the causal modeling tools leads to model formulations at the causal explanation level (see Section 1.1), although aggregated models can be defined by reusing predefined modeling components. The causal modeling component can be viewed as an *input/output processor* which predefines its physical reusing context with a given physical causality. As the electrical device in Figure 2.4 illustrates, different modeling components are required to represent the behaviour of the same physical device depending on the physical context. Model construction by means of reuse is therefore subjected to a previous analysis of the aggregated behaviour in order to determine which of the predefined modeling components gives a proper causal explanation to the reusing physical context. The connection topology of an aggregated model must preserve the adequate computational structure representing the aggregated behaviour. The resulting modeling component connections do not preserve the analogy with the system component connection topology. In consequence, causal modeling tools fails to fulfill with the automation model construction criterion postulated at Section 2.1 (the system specification can not be made at the topological structure level).

The behavioural modeling tools based on non causal equations (e.g. bond-graph (Thoma 1990) and equation-based object oriented (Cellier 1991)) support the system specification at the phenomenological structure level. Hence, most of the factors in the modeling process have been stated (phenomena of interest, adequacy and experimentation objectives). That is, the modeling component predefines its reusing physical context even causes and effects are not predefined. Hence, the modeller needs to analyze the physical context to decide whether the modeling component can be reused or not. The proper causal explanation of the aggregated behaviour is then obtained by means of the computational analysis of the equations. No automated support to select the modeling components can be given since the representation formalism does not have explicit information about the represented behaviour. An equation expresses how the system dynamics can be computed, but it has not meaningful information about the system behaviour. The analogy between the connection structure of predefined modeling components and the system topological structure can not be assured in the equation based object-oriented approach and is not preserved in the bond-graph approach. These modeling approaches have supposed a very important improvement with respect the causal modeling tools capabilities. Nevertheless, the automation model construction criterion is not fulfilled either.

With the mathematical formalism we can not go further into the higher levels of system specification (see Table 1.1). The possibility to specify the system at the phenomenological level is a consequence of the modeling methodology (object-oriented or bond-graph) and not of the representation formalism. Note that the phenomenological specification is built by reusing modeling components defined at the generative levels by means of a mathematical formalism (the difference with respect the causal tools is the interpretation of the equal operator as an equality relationship between math expressions). Hence, the system can not be specified at the topological level because the modeling components predefined at the generative levels determine their reusing physical context.

In order avoid this limitation and to accomplish with the objectives posed in Section 1.3, the PML language defines a dynamic modeling formalism which supports the system specification at the topological level. PML represents explicitly the physical knowledge required to translate the topological specification into the phenomenological specification suitable for the experimental framework.

The model usage criterion

This criterion embodies two aspects: the possibility to adapt a predefined model for different simulation purposes and the manipulations which will be required when the model does not fulfill with the simulator relation.

The simulation purpose sets the experimental framework and the desired adequacy level. By considering that both the causal approach and the behavioural equation approach are based on the static mathematical formalism, there are few capabilities to adapt a model in order to fit with the different experimentation goals, since the behaviour formulation at the generative levels must predefine which is the experimental framework and adequacy. In any case, there are no means to automate the possible adaptations. Some particular and application oriented exceptions may be found in some tools (e.g. model linearizing procedure in ACSL).

This is a limitation from the reusability point of view, thus it affects to the automation degree. We should consider that, in addition to the physical reusing context concerns, the model user must take into account his experimentation purpose in order to analyze and decide if a predefined model is valid or not.

The second essential aspect of every modeling methodology is the solution given to the model manipulation procedure required to translate the model specification into the formalism posed by the simulation engine.

In the causal approaches, obtaining a model formulation suitable for the simulation engine (numerical solvers) is a matter left to the modeller (an exception should be mentioned with certain CSSL tools which provide with equation sorting algorithms). That is, the modeling components must properly represent the predefined input/ouput causality and their connection must preserve an adequate computational structure. Therefore, if an algebraic problem (e.g. a set of simultaneous equations or high index DAE) appears when two or more modeling components are connected, the modeller has to manipulate the equations in order to eliminate it. If these requirements are fulfilled by the modeller, not very much work is left to the modeling tool in order to generate the simulation model because the mathematical structure of the causal models suits with the structure required by the numerical solvers.

The structured model formulations in the behavioural approach are not valid for numerical solution algorithms as *ODE* (Gustafsson 1992) or *DAE* (Brenan *et al.* 1989) solvers. The problem of achieving the model representation suitable for computer simulation is covered by means of

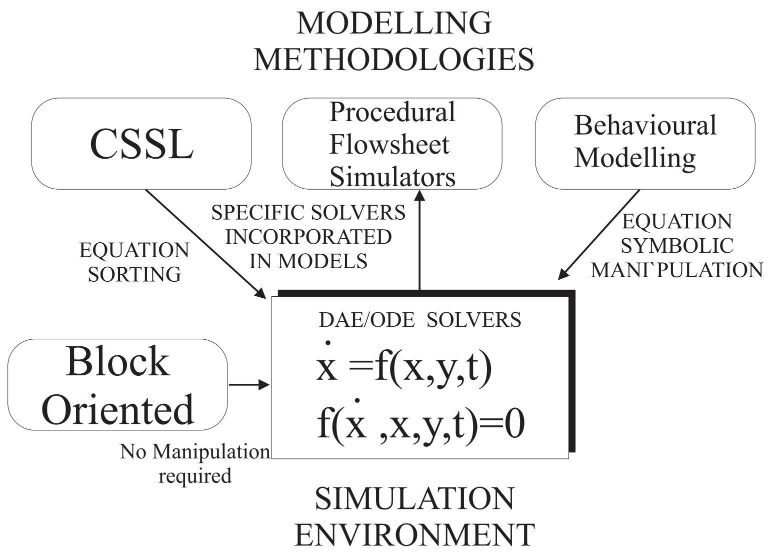


Figure 2.11. Manipulation procedures required to translate the model specification at the different modeling approaches into the causal explanation level where the simulator relation holds.

the computational analysis procedure (see Definition 1.6). The result of this procedure should be a proper translation of the non-causal formulation into the causal formulation. When in this translation procedure appear algebraic problems such as high index DAE or simultaneous system of non linear equation, the model user may be involved in their solution. This user intervention, related to the computational aspect of the model, must be avoided to consider automated the modeling process. Figure 2.11 summarizes the manipulation performed by the different modeling approaches in order to match the simulator relation.

2.3. Object Orientation within the physical modeling domain

As important as the representation formalism, the modeling methodology becomes the second main aspect to be considered in the design of the modeling environment. The object oriented paradigm, as design and implementation methodology, has been adopted in the modeling environment reported in thesis. This section is devoted to introduce the main characteristics and benefits of object orientation in order to understand the modeling methodology defined by PML.

Object technology is at its core the combination of four ideas (Meyer 1997): a *structuring* method, an *epistemological* principle, a *reliability discipline* and a *classification* technique.

As a structuring method, the object orientation is basically a way of solving a complex problem by dividing it in smaller problems which can be afforded independently and whose solution implementations can be reused in different contexts. The *class* is the representation structure which serves as the basis for this modular construction. The structured approach to build models consists in identifying a set of components and their structure of connections. The object oriented modeling approach offers a structuring method applying to model decomposition and reuse. Models of complex system are decomposed in a structure of ever simpler models. The *modeling class* becomes the mechanism, equivalent to the programming class, which supports the modular structure. Modeling classes can be reused by creating instances representing the corresponding system components. According to Definition 1.3, the modeling class is subsumed as the modeling component within the object-oriented modeling approach.

The *epistemological* principle addresses the question of how the modeling classes should be described. In object technology, the objects described by a class are defined by what can be done with them: operations (also known as *features*) and the formal properties of these operations (*contracts*). This principle has very much to do with the encapsulation feature. Classes have *attributes* to express the object properties and usually *methods* to operate on the attribute values. The information is encapsulated in the sense that the data structures (given by the attribute definition) are hidden and only accessible by carefully designed means: the *contracts.* Hence, the encapsulation becomes a technique to hide complexity and to make easy reusing predefined classes. The main implication within the physical modeling domain is that the modeling classes should provide the means to describe the physical behaviour (features) while offering the adequate interface defining the represented behaviour (contracts). Conceptually, any real or abstract entity (such as a electrical component or its mathematical model) can be considered as an object. The modeling class representing a physical object should express its behaviour, i.e., the object features (e.g., how current and voltage behave in the electrical component), and make the object accessible to other objects (other system components) by means of its interface (*contracts*). The interactions among the different objects can be established independently of the actual object feature implementation.

The *reliability discipline* is a radical approach to the problem of building software that does what is supposed to do (Meyer 1997). The idea is to treat any system as a collection of component which collaborate by adhering to *contracts* defining explicitly the obligations and benefits incumbent on each party. This technique is known as *aggregation* and applies to the physical structured modeling approach where a system is viewed as a set of interacting components (system specification at the structure levels). This notion of *composite* or *aggregate* model allows an explicit representation of decomposition structures. For instance, the system shown in Figure 2.3 is recursively structured in smaller subsystems. The smallest components are usually called *primitive* components since no aggregation is present. It is important to note that a composite object does explicitly refer to both its components and their connection structure.

Within the physical modeling domain, the way connections relate components likely involves complex physical interactions. For instance, a pipe connecting a chemical reactor and a heat exchanger involves a matter flow between these components, but also complex thermodynamic interactions will appear between the chemical reactor and the heat exchanger as the result of the matter transport. The aggregation technique success will depend on the capability of representing this interaction between objects, i.e., the object communication protocol. In object-oriented programming context, the communication is implemented by the *message passing* mechanism. However, in the physical modeling domain the communication between models will require a much more complex protocol in order to represent the physical interactions between the system components. As will be discussed in detail in chapter 3, the modeling class interface definition will be an essential point in the design of the physical behaviour representation formalism.

Finally, the object-oriented method relies heavily on a *classification* technique known as *inheritance*. All objects sharing the same attributes and methods are said to be *instances* of a *class*. Any (sub-)class can be derived from another (super-)class establishing an *inheritance* relationship. With inheritance, new attributes and methods (more knowledge) can be added to

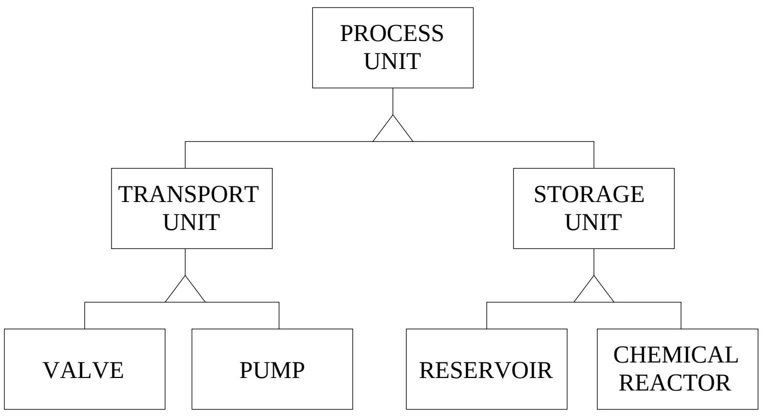


Figure 2.12. Chemical process units class hierarchy

(sub-)class definition, creating a class hierarchy by the specialization of the classes. To illustrate this concept look at Figure 2.12 where a taxonomy of chemical process elements are defined from the abstract class *Process Unit.* Inheritance is used to specialize the behaviour represented in a class. Each time we define a new class by means of inheritance, the behaviour represented at the superclass is incorporated into the subclass and new behaviour is added. For example, the `Valve` class at Figure 2.12 is used to represent the relationship between pressure drop and matter flow $Q = C_v A_p \sqrt{\Delta P}$, which specializes a more generic relationship $Q = \Phi(\Delta P)$ described at the `Transport Unit` class.

2.3.1. Quality factors of the Object Oriented method

This section introduces the principal benefits pursued by the object oriented method. Here follows a list of the most important quality factors, defined for the software construction domain (Meyer 1997) but applicable with slight adaptations to the modeling domain, whose pursuit is the central task of object oriented model construction:

Correctness is the ability of models to represent systems performing their exact task, as defined by their specification. The most usual model application will be based on simulation, so a proper imitation of the system behaviour is what it can be expected from a correct model. Usually, the approach to correctness is viewed in a *conditional* manner, since dif-

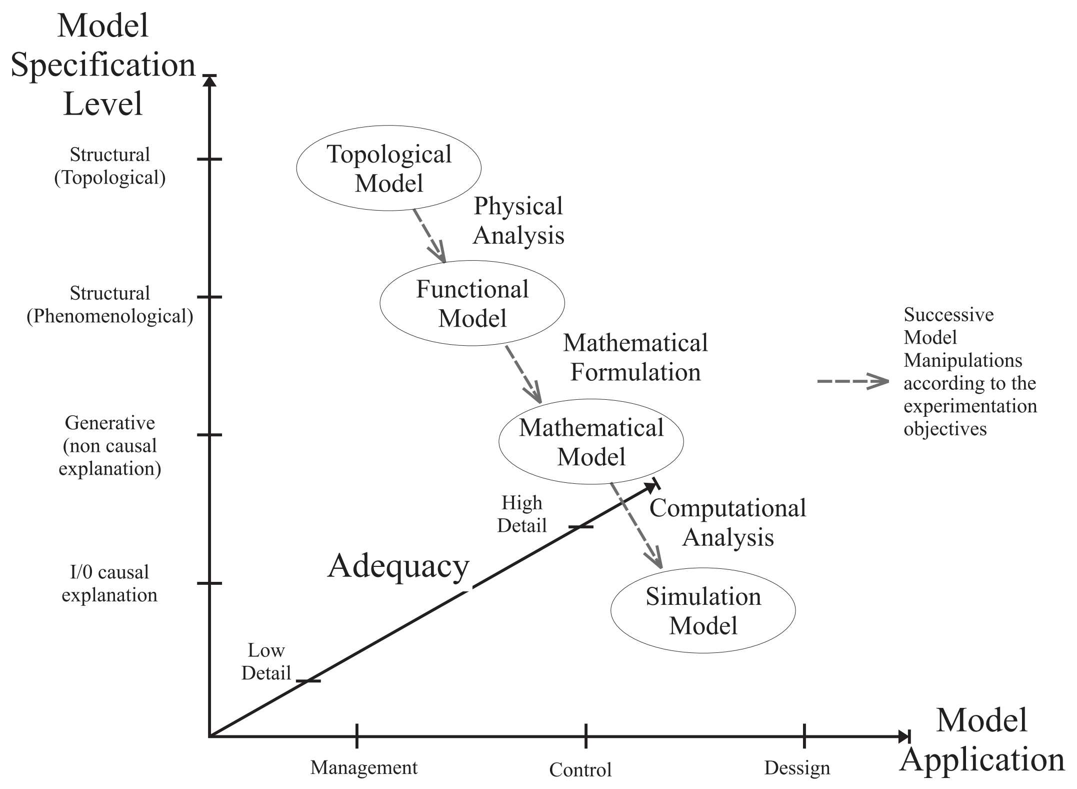


Figure 2.13. Layers in the generation of the simulation model: 1) construction of the topological model by means of reuse; 2) translation into the functional model by means of physical analysis; 3) mathematical formulation of the functional model; 4) generation of the simulation model by means of computational analysis.

ferent steps or layers are involved before the model can be operated. The conditional approach concentrates in guaranteeing that each layer is correct on the assumption that the lower levels are correct. A conditional method to ensure correctness becomes advisable in the modeling methodology presented in this work since, as the Figure 2.13 shows, multiple layers are involved.

Extendibility is the ease of adapting a model to changes of specification. This often will occur with models in different aspects according to the variety of their applications. The extendibility is a problem of scale and becomes relevant for complex models. Two principles are essential for improving extendibility:

- *Design simplicity*: the simpler architecture for building models the easier to adapt the model to changes. The capability of building structured models according to the architecture of the system will be shown as a simple approach suitable for the modeling purpose.
- *Decentralization*: the ability to circumscribe the effects of a change in a model will heavily depend in the autonomy of the modular structure which supports the model definition. Hence, this principle is closely related to the modular property that will be imposed to the modeling classes.

Reusability is the ability of modeling classes to serve for the construction of many different models. This is probably the most important mechanism for model construction and will play a central role in the discussions of this thesis. Reusability is a quality factor whose main benefit is probably the reduction of the model construction cost, but also it influences to improve other quality factors such as correctness. As it will be presented in the subsequent chapters, the nature of the reusable modeling classes depends on the formalism supported by the modeling language.

Compatibility can be defined as the ease of combining modeling classes with others. This is a very important factor considering the nature of a structured method. The modeling class compatibility will lie in the agreement of standardized conventions for model interaction.

Portability is the ease of transferring models to different modeling software environments. An ideal situation will be that component manufacturer could deal with the system components together with their models, so the client could evaluate the behaviour of the component within his process using his modeling environment. The portability factor goes one step beyond of compatibility and will be very related to the modeling class reusability.

Efficiency in the modeling domain means the ability of a model to place as few demands as possible on complexity considering a particular application for model experimentation. An ideal situation will be to have one unique model for a system and provide with the capability to adapt its mathematical formulation to the application purpose, simplifying this representation as much as possible according to the required complexity.

2.3.2. Main characteristics of Object Orientation

There is not a concise definition of object orientation, instead, a relation of its main characteristics is usually given. To decide whether a developing environment is object oriented or not is not a simple boolean choice. Here follows a list of properties that can be used as criteria to take this decision. This list does not pretend to be a dogmatic set of properties provided by an object oriented environment (a more exhaustive one can be found in (Meyer 1997)), however, it covers the fundamental concepts that should be supported. The current implementation of these rules in PML will be discussed in the Chapter 4.

Classification

The object oriented methodology is based in the concept of class. In software applications, the class can be informally viewed as an element representing an abstract data type and its partial or total implementation. An abstract data type is a set of objects defined by the list of operations (or *features*) applicable to these objects, and the properties of these operations. A *class* is a description of a group of objects with similar properties (Rumbaugh *et al.* 1991). The class declaration stresses the *"what is ..."* knowledge, rather than the *"how to ..."* knowledge. The class structure should comply with the following rules:

- The class should be a modular unit since object orientation is primarily an architectural technique whose main contribution is the modular construction of systems.
- The class is the type definition mechanism. The representation of every object in the application domain should be based on a class.
- Because of the modular nature of classes, the primary computational mechanism is: giving a certain object, which (because of previous rule) is an instance of some class, call a feature of that class on that object. This mechanism is also known as *message passing.* This *communication protocol* among objects is totally different in the system modeling domain.

```
modeling class Controller
  attributes:
   setPoint,inputSignal,outputControl;
  interface:
   setInputSignal(s)
   getOutputControl()
  behaviour:
   calculateControlLaw(setPoint,inputSignal)
end modeling class Controller
```

Listing 2.3. By means of *classification* the controller modeling class captures the properties common to a family of controllers: the attributes common to any SISO controller; the interface to set the controlled variable value and to get the control signal; an abstract method to declare the control law.

In system object oriented modeling applications, the concept of class is usually linked to the *model class* concept. Models can be regarded as classes in the sense that they capture the behaviour of a system type. A model class represents a family of physical objects or concepts sharing some common properties. For instance, the model class controller shown in Listing 2.3 can be considered as a representation of a family of *SISO* controllers with a set of common characteristics (e.g. sensor input signal, control output and the set point value) and whose control law realizations can be viewed as the implementation details of each particular controller. In this sense, a model class is a type definition and the modular property will be of principal fulfillment. However, there are significant differences between classes in programming and modeling. An important difference with respect software application is the communication between modules and its implications will be analyzed with detail in the Chapter 3.

Information hiding

In class definition two different facets should be distinguishable in the represented objects: the part which is accessible to other objects and the part which remains hidden and whose implementation details are irrelevant to the rest of interacting objects. This rule is known as *information hiding* and makes the class modular structure to be *encapsulated.* A main consequence of this rule is that communication between objects should be strictly limited. In the software domain, classes will exchange information exclusively through feature calls (message passing) and through the inheritance mechanism. In the modeling domain, because of the different translation procedures imposed by the simulation relation (see Section 2.2.3), the communication between the modeling classes will need a much more sophisticated manipulation. The model class interface will be the mechanism used to describe the communication between models, on the understanding that model communication usually represents physical relationships.

Genericity

For reasons of flexibility and extendibility, it should be possible to declare parameterized classes, known as *generic* classes. For instance, it should be possible to declare a *TREE* class operating with a generic type *T*, and then declare the classes which operate with specific types as *INTEGER* or *STRINGS*. Within the system modeling domain, a straightforward counterpart follows: it should be possible to declare a modeling class representing a generic *TANK* storing *MATTER*, and then to declare specific modeling classes with specific modeling classes such as *WATER* or *ETHANOL*.

Inheritance

This is a powerful mechanism for sharing similarities among classes while preserving their differences. Inheritance is the relationship between a refined class and a more general class. A class will be an heir of another if it incorporates the other's features in addition to its own. For example, a `PidController` modeling class refines the more generic `Controller` modeling class (see Listing 2.3) by declaring the specific behaviour of the PID control law. In turn, the `Controller` modeling class generalizes the PID controller. Generalization and inheritance are transitive across an arbitrary number of levels.

An object oriented environment may support single inheritance and multiple inheritance (e.g. a *car* class may inherit both from the *engineered vehicle* class and the *four-wheels vehicle* class). The PML language only supports single inheritance nowadays.

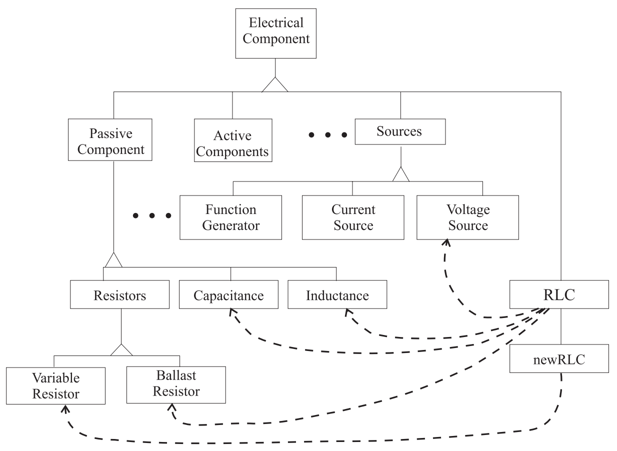


Figure 2.14. *Polymorphism* extends composite model reusing by allowing the exchange of polymorphic model classes. A new RCL model class can be defined by exchanging the polymorphic `Ballast Resistor` and `Variable Resistor` modeling classes. Continuous lines represent *heir class* ↔ *ancestor class* relationships (inheritance). Dashed lines represent *aggregation* relationships.

Redefinition

When a class is a heir of another, it may need to redefine some of the inherited features. Model classes represent the system behaviour. The redefinition mechanism means that part of the inherited behaviour may be changed affecting to the represented behaviour. For instance, a reservoir system model containing some liquid may be redefined to describe the storage of gas, adapting the representation to the new type of medium while preserving the common features.

Polymorphism

In software systems, *polymorphism* is the ability for an entity (names representing run-time objects) to be attached to run-time objects of different types. Polymorphism is not arbitrary and it is under the inheritance control. For instance, features of class *C* can be attached to objects of descendant classes of it with polymorphism. Then, the same feature may behave distinctly with objects of different classes.

The polymorphism concept in the system modeling domain needs an explanation. Basically, two model classes are polymorphic if they can be used without distinction in the same context. For example, the Figure 2.14 shows how a new *RLC* circuit model can be defined by the substitution of a model component in its ancestor *RLC* circuit model. If the components can be indistinctly exchanged are said to be polymorphic. The demands on polymorphic models will depend on the formalism provided by the modeling language, and will be discussed with detail in Chapter 4.

Dynamic binding

Can be seen as a combination of the two previous mechanisms. Assume a call whose target is a polymorphic entity, for instance a call to a feature of an entity declared of type *C*. The various descendant of *C* may have redefined the feature in various ways. There must be an automatic mechanism to guarantee that the called feature version correspond to the actual object type. This property is named as *dynamic binding*.

In the modeling context, the dynamic binding is the ability to adapt automatically the behaviour representation to the physical context. For instance, exchanging the liquid medium by the gas medium in the reservoir system model will require a different representation according to the change in the physical behaviour. This capability is an important contribution of PML, and the details will be presented in the Chapter 4.

Libraries

One the characteristic aspects of the object oriented methodology is the ability to rely in libraries. An object oriented environment should provide basic libraries containing the basic class definitions and the mechanism to write more reusable classes.

2.3.3. The Object Oriented modeling method

This section is devoted to analyze relevant aspects to be considered when the object-oriented methodology is moved to the modeling domain. The reasons why the object oriented method has been adopted in the system modeling domain are analogous to other engineering domains such as software. The two main goals are to minimize model building effort and cost while obtaining reliable models. In analogy with software development, the main approach consist in supporting a modular architecture and module reusability. When already defined and validated models can be reused for building new models, the development cost can be significantly reduced.

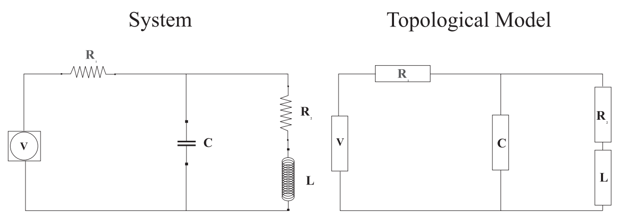


Figure 2.15. System $\Longleftrightarrow$ Model structural analogy.

Despite the analogies with object oriented programming, there are several aspects which are particular to the physical system modeling domain. These singularities are mainly due to the following aspects:

- Reuse of predefined modeling classes. The behaviour of a system depends of the context where it is operated. The motor drive presented previously is an example of how the operation mode affects to the mathematical formulation of the system dynamics. Hence, a modeling environment supports reusability if it has the ability to formulate the adequate behaviour.
- Topological system specification. The model of a system can be decomposed in many different ways. What it should be possible is to make this decomposition according to the system structure in order to ease the model building task. If a system can be viewed as a collection of independent subsystems assembled according to certain architecture, it would be desirable that a model could be described as collection of independent submodels assembled in the same way (see Figure 2.15).
- Constraints imposed by numerical solvers. The mathematical formulation of a model must suit the formalism imposed by numerical solvers (the simulator relation), so the modeling environment should be able to adequate structured models to these formalism (e.g. ODE or DAE mathematical forms).

These aspects give a specific meaning to the following properties that should be accomplished to provide an object oriented modeling methodology:

- Declarative representation of system behaviour.
- Representation supported by modular structures.
- Model construction by Inheritance an Aggregation.

Declarative representation of behaviour

The numerical solvers used in computer simulation impose to the model mathematical formulation to be in the ODE form :

$$\frac{\partial x}{\partial t} = f(x, u, t) \tag{2.24}$$

or DAE form:

$$f(\frac{\partial x}{\partial t}, x, u, t) = 0 \tag{2.25}$$

where t is the time, x is the state vector , u is the input vector and y is the ouput vector.

This causal *input/state/ouput* representation brought about the modeling mechanisms in the classical simulation environments, such as *ACSL* (Mitchel and Associates 1986) o *Simnon* (Elmqvist *et al.* 1990). The essence of these mechanism is the procedural representation of the system behaviour. It has been already analyzed which are its limitations from the model reusability point of view. One of the characteristics required to guarantee reusability is to have declarative behaviour representations: modeling classes should be declarative in the sense that they define physical behaviour and knowledge, rather than being procedures for computing data.

Since computer simulation requires causal formulations according to a given structure, declarative modeling classes are not directly applicable. Hence, the modeling tool must be able to perform the proper translation procedure by analyzing the model reusing context.

How declarative knowledge is translated into the procedural formulation required by numerical solvers will depend on the formalism used to represent the behaviour. For instance, the object oriented modeling languages based on behavioural equations interpret the operator '=' as the symmetric equivalence operator. For example, in a resistor model, defined by Ohm's law, the equations $V = I * R$ or $I = V/R$ are equivalent. The proper assignment may be stated once the model reusing context is established.

The need to adapt the model declaration to a proper formulation for numerical solvers is particular to the modeling domain and establishes a difference with respect other object oriented disciplines.

Modularity

Modularity. A module is a group of entities which are in some way related. A module must specify the input/output coupling as well as the internal entity coupling. A model representation must support modularity at multiple levels. By means of modularity, a model may have submodels which have submodels themselves. This gives a hierarchical decomposition which can be arbitrarily deep. A model which have submodels is called a *composite* or *structured* model, otherwise it is called *atomic* model.

Abstract interfaces. This concept is very related with modularity. Interfaces define the interaction of a module with its environment. Abstraction consists on focusing on the essential aspects of this interaction relationships. For instance, a pipe model interface may be defined by the variables (pressure and mass flow) shared with its environment, or may be defined by a more abstract declaration of matter exchange. Use of abstraction preserves the freedom to make decisions as long as possible by avoiding premature commitments to details (Rumbaugh *et al.* 1991). Modularity and abstraction are closely related to *encapsulation* (information hiding). This means that the entities declared in a module can only be accessed through module interface.

2.4. Overview of PML

At this point we have already presented a deep insight of the main goal of the work reported in this thesis: the design of a modeling environment able to automate the modeling process as it is described in Chapter 1. In relation to the modeling process automation, we have analyzed the main limitations of the most wide spread modeling approaches. This analysis serves us to realize about the key points to be considered in the design of a modeling environment able to overcome the highlighted limitations. Here follows the description of the main ideas behind the PML design. They will be fully developed in Chapter 3, 4 and 5.

We should distinguish, as in any other approach, two main elements in the PML modeling environment: the modeling methodology and the representation formalism. Because of the reasons exposed in Section 2.3, we have adopted the object-oriented approach as the paradigm to be supported by PML, trying to maximize the benefits of the approach when it is applied within the system modeling domain. In difference with other object-oriented modeling tools, the representation formalism is not based on equations. We have already analyzed which are the limitations of the equation as a modeling formalism: it is needed to represent the system dynamics but is not useful to represent the system behaviour and the physical knowledge required to analyze it. The PML language defines a dynamic formalism to represent the physical behaviour and the physical knowledge required to analyze it (see Definition 1.8).

The PML modeling methodology pretends to combine the flexibility of the object-oriented modeling approach and the ease-of-use of the block-oriented modeling approach: a complex model of (ideally) any system may be configured just by selection, parameterization and aggregation of modeling classes retrieved from a modeling library. PML has been designed to achieve the following major objectives:

1. The system can be specified at the topological level of the modeling process described in Chapter 1. This specification is named topological model (see Definition 1.9). The construction of the topological model is based on modeling classes which do not predefine their reusing context. Making the reusability context independent includes the fulfillment of two features: to obtain the aggregated behaviour from the physical analysis of the topological model structure and manipulate it according the experimental framework defined by the model user. This leads to the system specification at the phenomenological structure level named functional model (see Definition 1.10).

2. Keep track of user modeling purpose by offering the proper information required to establish all the hypothesis, assumptions and simplifications that give rise to a particular simulation model. The physical behaviour represented by the functional model can be translated into the non causal explanation level taking into account the adequacy factor. This means that the system dynamics can be formulated at different detail levels according to the model user simulation purposes.

3. The topological model is built with a modeling language which articulates the physical behaviour representation in terms of concepts familiar to engineers. This representation is uncoupled from the mathematical formalisms required for computer simulation.

4. To automate the modeling trajectory from the topological model to the simulation model. This includes shielding the user from any problem through this trajectory not related with the modeling activity.

In order to fit with these major objectives, we should found the object-oriented approach on a formalism suitable for the physical behaviour representation. Equation based environments, such as declarative flowsheeting and generic modeling languages based on the object oriented paradigm (e.g. Modelica, EcosimPro), support the implementations of models to a large extent, but they do not assist the user in developing models from using engineering concepts, nor there is support for the documentation of the modeling process during the life cycle of a process or for a proper design and documentation of the model library. Reuse of validated unit models by a group of simulator users is therefore almost impossible and redundant modeling is unavoidable (Marquardt 1991). The equation based formalism presents several limitations in order to support a complete unfolding of the object-oriented methodology. The chapter 3 is devoted to analyze the need of the PML representation formalism in order to overcome the limitations of the equation-based formalisms. The need of this new representation formalism has been also reported in previous works such as (Ramos 1995, Ramos *et al.* 1998*a*, Ramos *et al.* 1998*b*, Piera *et al.* 1998, Ramos and Piera 1999)

The representation formalism defined by the PML language separates the behavioural aspects and the computational aspects. According to Definition 2.1, a PML model will describe the behaviour $\mathcal{B}$ in terms of the declaration of the phenomena happening in the system. The physical behaviour is represented with new semantic constructs which are closer to the behavioural system description than equations are. PML language constructs such as phenomena and physic laws have a semantic role themselves and are near to the engineer's understanding. The equation formalism is used in the PML modeling environment to write down the mathematical model (see Definition 1.11). In other words, equations are used to build the system specification at the non causal explanation level. Hence, the equation formalism does not play any semantic role in PML, it is merely the vehicle to formulate the system dynamics according to the experimental framework and adequacy level. The following example illustrates the behaviour representation structures (they are fully developed in Chapter 4).

EXAMPLE 2.2 – *Physical model of a reservoir system*

Using a dynamic formalism allowing the explicit representation of physical concepts, the tank model which was discussed at Example 1.4 could be defined as a record of the represented physical phenomena. Then, by emulating an engineer way of proceeding, the tank model could be defined with an expression such as:

$$\texttt{the tank model represents the matter storage} \tag{2.26}$$

This behavioural description of the tank system can be related with the behaviour mathematical representation through the physical law which rules the storage phenomenon:

$$\texttt{store(matter)} \longrightarrow \texttt{massBalance(matter)} \tag{2.27}$$

Finally, the mathematical formulation is linked to the law representation.

$$\texttt{massBalance(matter)} \longrightarrow \frac{\texttt{dm}}{\texttt{dt}} = \sum_{\texttt{i=1}}^{\texttt{n}} \texttt{w}_{\texttt{i}} \tag{2.28}$$

This way of organizing the physical knowledge leads to a mathematical formulation analogous to Equation 1.8, but also establish an explicit representation of the physical knowledge with a much more natural expressiveness. We would proceed in a similar way with the valve model.

To the reusability concern, minimum interpretations have to be done in order to restore the represented behaviour to consider how the model can be reused. Furthermore, no assumptions about the reusing context are made in the model representation of Expression 2.26. For instance, the use of this tank model can be extended to model other tank systems where different materials, single or multi-component, are involved. It is just a question of stating the proper behavioural equation in each case. ❑

The essential requirements which the PML language satisfies are:

(i) The topological model articulates declarative knowledge in such a way that the represented behaviour reuse is context free.

(ii) The declarative knowledge (behavioural concerns) is completely uncoupled from the procedural knowledge (computational concerns) by means of modular structures which encapsulate the physical behaviour representation.

(iii) The language is able to afford generic-domain modeling problems where heterogeneous components from different engineering domains are involved.

The explicit representation of the physical behaviour in terms of the phenomena occurring in the system will permit to design modeling classes without predefining their reusing context. The topological model is built by aggregating the predefined modeling classes. The proper coupled behaviour is obtained from the physical analysis of the topological model. This analysis is based on the physical behaviour and knowledge explicitly represented by the language constructs. This behaviour declaration will be formulated in terms of behavioural equations by setting the laws ruling the represented phenomena. The mathematical representation of the system behaviour is formulated once the reusing context of each model and the resulting coupled behaviour have been established. From this point of view, we may consider that the behaviour mathematical representation is *dynamically* formulated from the physical analysis of the model.

One important consequence of the separation between the behavioural and computational concerns is the differentiation between the modeling and the simulation activities achieved by PML. It will contribute to increase the model reusability by means of the adaptation of the system topological model in order to give answers to different experimentation demands and objectives. This adaptation will require of model manipulations where such demands are taken into account. Other modeling approaches, stemming from the artificial intelligence domain such as the HPT (Woods 1993) or the modeling approach in (Nayak 1995), do also advocate the convenience of providing the user with models that can be adapted for different experimentation goals.

The formalism used to represent the physical knowledge will establish to a great extend the boundaries to the capability of assisting the user in these modeling tasks. Model representations based on variables and equations can only be manipulated within the mathematical formalism context. Outside this context, no support can be provided to the user. On the contrary, manipulations within the physics context can be expected from a formalism specific for the physical knowledge representation. With the explicit knowledge formulation provided by PML, three main issues can be achieved:

i) The user has an expression mechanism (language) close to his way of thinking in physical concepts, getting clear view of the behaviour represented in a model.

ii) The modeling tool designed to support such a language can manipulate the models from a physical point of view, adapting the model formulation to its reusing context. This formulation takes into account the experimentation purpose.

iii) The simulation model can be automatically derived from the represented physical behaviour.

These issues would provide a higher level analysis capability, in the sense of applying to physical

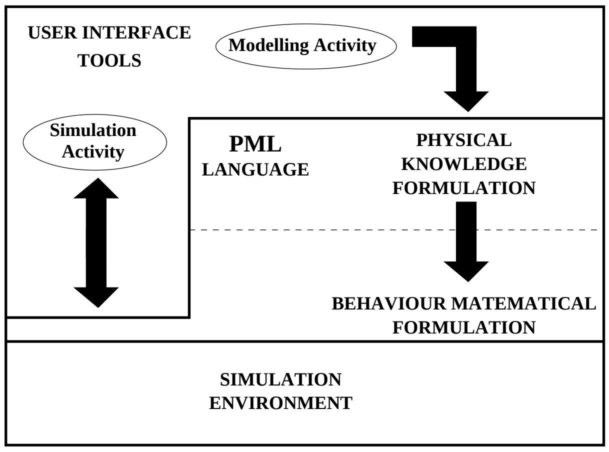


Figure 2.16. The PML modeling and simulation framework. Modeling and Simulation activities have been clearly separated. The modeling activity is supported by means of an automated translation of the topological model into the simulation model. Each step in this trajectory remains hidden to the model user.

knowledge, than the mathematical formalism provides. The PML language has been designed to support these features. As it is shown in the Figure 2.16, the PML language is conceived as a physical knowledge representation layer on top of the system behaviour mathematical formulation. PML provides a representation vehicle close to the physical concerns of the system in order to make a clear separation between the modeling activity, as a system representation task, and the simulation activity, as a computational aspect of the model. The PML formalism allows models to be expressed in terms of physical behaviour (chapters 3 and 4 are devoted to analyze the context where this formalism emerges and to present how the physical knowledge is organized and represented). Such a model definition is then analyzed from a physical point of view by the modeling tool in order to obtain the mathematical formulation suitable for numerical simulation. The user interacts with the modeling environment through software interfaces which assist him in the manipulation of the models. The modeling environment takes care of the mathematical formulation of the model in order to interact with the simulation tools. Therefore, the modeling environment designed around PML can be considered as a front-end between the user's modeling concerns and the computer's simulation concerns.